1 Maximum Flows and Minimum Cuts

In this lecture, we'll learn about max-flow algorithms, their relationship to a graph's *minimum cut*, and a fundamental result from graph theory called the max-flow/min-cut theorem. Max-flow algorithms are a common technique for solving a wide variety of problems, including assignment or matching problems.

In max-flow, we are given as input (i) a directed graph $G = (V, E)$, (ii) an edge weight function $c : e \to \mathbb{R}_{\geq 0}$ that represents the *capacities* of edges $e \in E$, and (iii) a pair of vertices $s, t \in V$ that represent the *source* and *target*. The output is a *flow* function $f : e \to \mathbb{R}_{\geq 0}$ with maximum magnitude $|f|$, subject to the constraints the flow is *conserved* and that f is *feasible*.

1.1 Flows, conservation, and feasibility

The conservation constraint on f requires that the flow *in* to some vertex v equals the flow *out* of that vertex v. Mathematically, conservation means

$$\forall_{v \neq s,t} \qquad \sum_u f(u \to v) = \sum_w f(v \to w) \ ,$$

where we say that $f(u \to v) = 0$ if $(u, v) \notin E$. A useful analogy is to think of the graph G as a set of pipes connected together in a network, with water flowing from the source s through the pipes to a drain at the target t.

The conservation constraint applies to every $v \neq s, t$. Flow can only be created at the source s (out-flow exceeds in-flow) and flow can only be consumed at the target t (in-flow exceeds out-flow). Given a choice of s and t, a flow f can be called an (s, t)-flow on G. (If we change the choice of s or t, the maximum flow possible on G may also change.) The value of a flow f is defined as

$$\begin{aligned} |f| &= \sum_v f(s \to v) - \sum_u f(u \to s) \\ &= \sum_v f(v \to t) - \sum_u f(t \to u) \ . \end{aligned}$$

That is, $|f|$ equals the net out-flow at s which equals the net in-flow at t. (Do you see why?)

Feasible flows. A flow is *feasible* with respect to the edge capacities c if for every edge $e \in E$, the capacity of e is not exceeded by the flow on e, i.e., we require that $f(e) \leq c(e)$. Because the capacity function c is fixed on the input, we only consider feasible flows with respect to the given c. Continuing our analogy with water and pipes, a large pipe has a large capacity, and because water is an incompressible fluid, if a pipe has a capacity of k units of water flow, no flow through G can push more than k units through that pipe. Trust me, no one wants broken pipes.

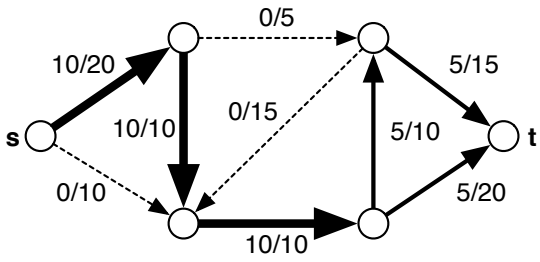


Figure 1: A graph G and a feasible (s,t)-flow with value $|f| = 10$. Edge labels give the flow divided by the capacity. The flow f avoids the dashed edges and saturates exactly two edges.

The maximum flow problem. *Given a graph G, capacity function c, and choices s, t, compute a feasible (s,t)-flow on G whose value $|f|$ is maximized.*

1.2 Cuts

A *cut* on G is a bipartitioning of the vertices V into subsets S and T (meaning $S \cup T = V$ and $S \cap T = \{\}$), with $s \in S$ and $t \in T$. We call such a cut an (s,t)-cut. (How many (s,t)-cuts are there for a graph $G = (V, E)$?)

Given an (s,t)-cut and a capacity function c (as above), the *capacity* of the cut is the total capacity of the edges "in" the cut, which we define as edges that start in S and end in T. Mathematically, this means

$$||S,T|| = \sum_{v \in S} \sum_{w \in T} c(v \to w) \ .$$

The capacity of $||S,T||$ is not necessarily equal to the capacity of $||T,S||$. (Do you see why not?)

The minimum cut problem. *Given a graph G, a capacity function c, and choices s, t, find an (s,t)-cut whose capacity $||S,T||$ is minimized.*

As a result, the edges in this cut are thus the minimum-cost edges required to disconnect s and t.

1.3 Max-Flow Min-Cut Theorem

Saturated and avoided edges. There are two limiting cases for how a flow f uses an edge e. If $f(e) = c(e)$, i.e., f maximizes the flow on e, we say that f *saturates* the edge e. And, if $f(e) = 0$, i.e., f places the minimum flow on e, we say f *avoids* the edge e. These limiting cases will be useful a bit later, when we show the connection between max-flow and min-cut.

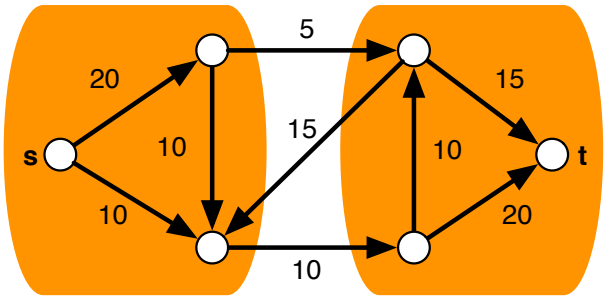


Figure 2: The same graph from Figure 1, now showing an (s,t)-cut with capacity $||S,T|| = 15$, which is also the minimum cut. Edge labels give the capacity only. In this case, $||S,T|| = ||T,S||$, but the graph would need to be modified only slightly to make this not true.

The equality condition. We can prove that the value of any feasible (s,t)-flow is at most the capacity of any (s,t)-cut. In fact, however, an even stronger statement holds, which is called the *equality condition*: the value of a flow $|f| = ||S,T||$ if and only if the flow f saturates every edge from S to T and avoids every edge from T to S. (Why is avoiding edges from T to S important?) Moreover, for a flow f and an (s,t)-cut that satisfies this equality condition, f must be a maximum flow and the (s,t)-cut must be a minimum cut.

For any weighted directed graph, there is always some flow f and some (s,t)-cut that satisfy the equality condition. Stated more precisely:

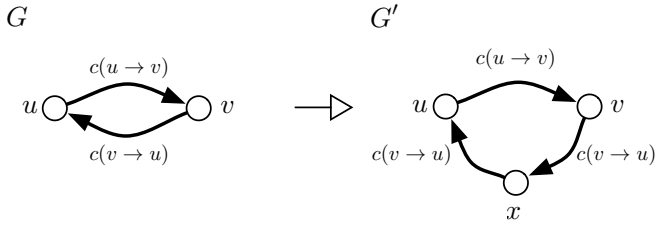
Max-Flow Min-Cut Theorem: The value of the max flow equals the capacity of the min cut.

The proof here is constructive, meaning that it yields an algorithm that we can use to find the maximum flow.[1] The proof exploits a transformation of the input graph G into a residual capacity graph G_f, where it is easier to find the maximum flow than it is directly on G. This approach works because we guarantee that the maximum flow on G_f is the maximum flow on G.

To construct G_f, we begin with a fixed, directed graph $G = (V, E)$, vertices $s, t \in V$, and a capacity function c. The first step is to convert G into a *reduced graph* $G' = (V', E')$. In G', we require that for all pairs of vertices u and v, either $c(u \to v) = 0$ or $c(v \to u) = 0$, i.e., if an edge appears in G', then its reversal does not. This transformation will allow us to use an edge (u, v) and its reversal (v, u) in G_f to store the residual capacity and flow on (u, v) in G.

[1]The max-flow min-cut theorem was first proved by Lester Ford and Delbert Ferguson in 1954, who gave the first algorithm for solving it, and then independently by Peter Elias, Amiel Feinstein, and Claude Shannon in 1956.

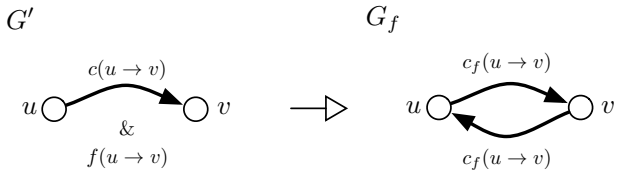
To deriving the reduced graph G', which contains no bidirectional arcs, we take each pair of non-reduced vertices $\{(u,v),(v,u)\}$ and replace them with a small subgraph $\{(u,v),(v,x),(x,u)\}$, where $c(v \to x) = c(x \to u) = c(v \to u)$. This inserts a new vertex x into one of the two offending edges and thereby eliminates the redundancy. Because flow is conserved, the max flow of G' will be the same as G (do you see why?).



Finally, given G' and a feasible flow f, we can then construct the *residual graph* $G_f = (V, E_f)$. We define the *residual capacity* edge weight function for G_f as $c_f : V \times V \to \mathbb{R}_{\geq 0}$. This function keeps track of both the used and remaining capacity on an edge, given a feasible flow f:

$$c_f(u \to v) = \begin{cases} c(u \to v) - f(u \to v) & \text{if } (u,v) \in E' \\ f(u \to v) & \text{if } (v,u) \in E' \\ 0 & \text{otherwise .} \end{cases}$$

If an edge (u,v) appeared in the original graph G, then in the residual graph G_f, the edge (u,v) is given a weight equal to the residual capacity on $(u,v) \in E$ under f, while the edge (v,u) is assigned a weight equal to the flow on $(u,v) \in E$ in f. Because $0 \leq f(u \to v) \leq c(u \to v)$, residual capacities are always non-negative.



The edge set E_f of the residual graph is defined as the set of edges whose residual capacity under flow f is positive; these are edges in G that are not saturated. By storing the residual capacities on the edges in E, a path from s to t on the residual graph G_f implies the existence of some non-zero amount of additional flow that can be added to f. Hence, transforming G into G_f makes it easy both to find incremental increases to the total flow, and keep track of which edges are saturated and which have some unused capacity available.

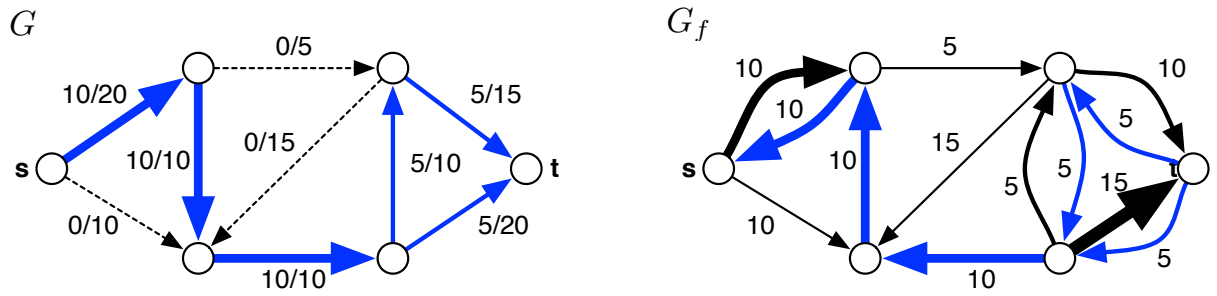


Figure 3: The same f and G as before, now with the corresponding residual graph G_f. Blue edges represent flows, which point "forward" in G ($u \to v$) and "backward" in G_f ($v \to u$). Black edges represent residual capacity for $u \to v$. Can you spot an augmenting path in G_f?

No s,t path in G_f implies f is a max flow. Suppose there is no path from s to t in the residual graph G_f. Let S be the set of all vertices that are reachable in G_f from s, and let $T = V - S$ (the rest of the graph). The partition (S,T) is then an (s,t)-cut, and for every pair of vertices $u \in S$, $v \in T$ (the edges in the cut), we have

$$c_f(u \to v) = \Big(c(u \to v) - f(u \to v)\Big) + f(v \to u) = 0 \ .$$

This fact implies that (i) every edge $S \to T$ is saturated (i.e., $c(u \to v) - f(u \to v) = 0$) and (ii) every edge $T \to S$ is avoided (i.e., $f(v \to u) = 0$). Thus, f is a max flow and (S,T) is a min cut.

Augmenting paths in G_f. Suppose there exists an s,t path $\sigma_{st} = \{s = v_0 \to v_1 \to \cdots \to v_r = t\}$ in G_f. Call σ_{st} an *augmenting path*. It follows that there is some non-zero residual capacity

$$F = \min_i c_f(v_i \to v_{i+1}) > 0$$

available on every edge in this path. Hence, we can increase the value of f by pushing F additional units of flow through G along the path defined by σ_{st}, and f was not a maximum flow. Given that we have identified some σ_{st}, all that remains is the accounting of updating the flow f to add F along the edges in σ_{st}.

Each edge $(u,v) \in \sigma_{st}$ is either a "forward" edge, meaning that it represents some unused capacity, or a "backward" edge, meaning that it represents some current flow on the edge (u,v) in the original graph G. The definition of an augmenting path allows it to traverse either type of edge in the residual graph. If the path traverses a forward edge, then we decrease its residual capacity by F and increase the flow on the corresponding backward edge. Or, if the path traverses a backward edge,

then we increase its residual capacity by F and decrease the flow on the corresponding forward edge.[2]

Given an augmenting path with minimum residual capacity F, we update the flows on G using each edge $(u, v) \in \sigma_{st}$. Let f' denote a new flow on G, defined as

$$f'(u \to v) = \begin{cases} f(u \to v) + F & \text{if } (u, v) \in \sigma_{st} \qquad \text{(a forward edge)} \\ f(u \to v) - F & \text{if } (v, u) \in \sigma_{st} \qquad \text{(a backward edge)} \\ f(u \to v) & \text{otherwise .} \end{cases}$$

It is crucial to notice two things: (i) this update function applies to the flows f and not to the residual weights c_f, and (ii) it is applied once for every edge in the augmenting path.[3] If the augmenting path traverses an edge in G_f in the "forward" direction, i.e., an edge with non-zero residual capacity, then we *increase* the flow $f(u \to v)$ by F (and hence decrease the residual capacity $c_f(u \to v)$). Similarly, if the augmenting path traverses an edge in G_f in the "backward" direction, i.e., an edge with non-zero flow, then we *decrease* the flow $f(u \to v)$ (and thereby increase the residual capacity $c_f(u \to v)$).

We now prove that the updated flow f' is still a feasible flow with respect to the original capacities c, by verifying that $0 \leq f' \leq c$ for each edge, i.e., we haven't burst any pipes. Since we only modify edges in the augmenting path itself, we must only check whether we have violated any of their capacities. If the forward edge $(u, v) \in \sigma_{st}$, then we increase its flow and $f'(u \to v) > f(u \to v) \geq 0$ and thus

$$\begin{aligned} f'(u \to v) &= f(u \to v) + F \\ &\leq f(u \to v) + c_f(u \to v) \\ &= f(u \to v) + c(u \to v) - f(u \to v) \\ &= c(u \to v) \end{aligned}$$

On the other hand, if the backward edge $(v, u) \in \sigma_{st}$, then we decrease its flow and $f'(u \to v) <$

[2]The idea of removing flow from (u, v) may seem counter-intuitive. Notice that the only way σ_{st} can traverse a backward edge is if we had previously found an augmenting path that traversed it in the forward direction, which would have added some flow in the backward direction in G_f. W.o.l.g. we can say that previous augmenting path had the form {some s, v path, (v, u), some u, t path}. The only way (u, v) can be in σ_{st} is if σ_{st} has the form {some s, u path, (u, v), some v, t path}. Because we add F to all the edges in the s, u and v, t paths, we must add F flow to some edges (x, u) and (v, y), at the same time we remove F from (u, v). Hence, removing flow from (u, v) does not violate the conservation constraint, and will in fact increase the flow by F. Intuitively, this kind of update takes the first part of the old augmenting path and stitches it onto the second part of the new augmenting path, and vice versa, to produce two paths with net more flow than the old path.

[3]Either $(u, v) \in \sigma_{st}$ or $(v, u) \in \sigma_{st}$, but not both. Do you see why?

$f(u \to v) \leq c(u \to v)$ and thus

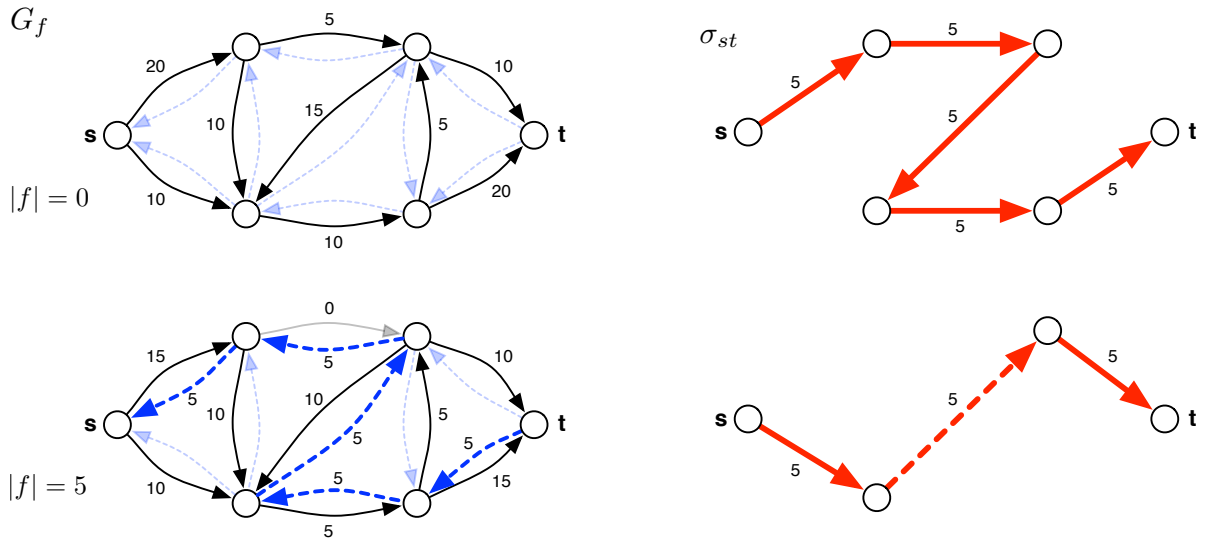
$$\begin{aligned} f'(u \to v) &= f(u \to v) - F \\ &\geq f(u \to v) - c_f(u \to v) \\ &= f(u \to v) - f(u \to v) \\ &= 0 \end{aligned}$$

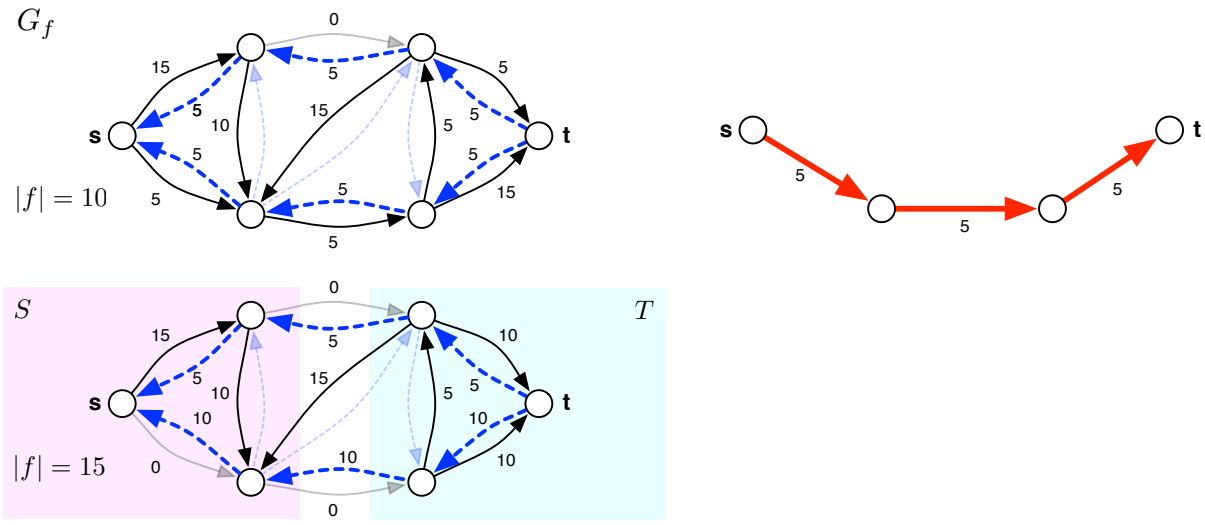
Finally, since $|f'| = |f| + F > 0$, f was not a maximum flow. □

Thus, we have proved that if f is a maximum flow, there will exist no augmenting path in G_f and (S, T) is a minimum cut. The neat thing about this proof is that it is constructive, i.e., it not only proves that the maximum flow is the minimum cut, it tells us how to find it: repeatedly find an augmenting path in G_f until no such path exists.

1.4 A worked example

Consider again our example graph G from above, but now we initialize f to be an empty flow. The following figures show a sequence of residual graphs G_f and an augmenting path on that G_f. Each time we identify an augmenting path, we apply the update function to change the flow f, and hence change the residual capacity weights on G_f to increase the value of the flow. The second augmenting path shows the operation of moving flow off an edge, as the augmenting path follows a "backward" edge.

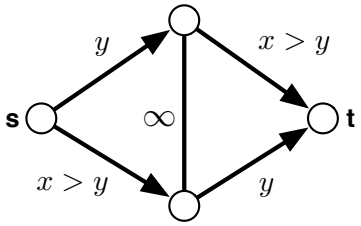




The final G_f showing both the maximum flow of $|f| = 15$ and the minimum cut $||S, T|| = 15$, where every edge from $S \to T$ is saturated and every edge from $T \to S$ is avoided.

1.5 A simple greedy approach fails

The max-flow problem is a classic example for how a naïve greedy algorithm can fail to find the optimum solution. For the max-flow problem, a simple greedy approach would be to (recursively) open (saturate) the fattest pipe first (what we might call "OFPF"). That is, start with the original G graph, and saturate the largest capacity path from s to t first, and then recurse until no more flow can be pushed through. (Note that this approach is similar to the idea described above, except that above we push flow on the residual graph G_f, not the original graph G.)



To see how this can fail, consider the above graph G, in which edges have capacities of x, y or ∞, and where we assume $x > y$. The OFPF greedy solution is $2y$, while the optimum, which a greedy approach on the residual graph will find, is $x + y > 2y$.

1.6 Ford-Fulkerson algorithm

All of the max-flow min-cut algorithms in this lecture work by finding an augmenting path in the residual graph G_f, but they differ in how they find it. The simplest approach is to just find *some* augmenting path, without caring about which one we find.

This approach is the idea of the Ford-Fulkerson algorithm (1954), which was the original solution to the max-flow problem: start with an empty flow $f = 0$ and then repeatedly use DFS to find some augmenting path in G_f, each time updating f to include the corresponding additional flow, until no s, t path can be found.

Pseudocode for Ford-Fulkerson looks like this:

```
FordFulkerson(Gf) :
   for all edges: f(u,v) = 0                  // an initially empty flow
   sigma = DFS(Gf,s,t)                        // run DFS from s to t
   while sigma not empty :
      F = minimum residual capacity of sigma  // amount to add to f
      for each (u,v) in sigma :               // update f for each edge in sigma
         f(u,v) = f(u,v) + F                  //   add F flow along (u,v)
         f(v,u) = f(v,u) - F                  //   remove F residual capacity from (u,v)
```

If all the edge capacities are integers, then each augmentation step increases $|f|$ by a positive integer which is at least 1. Thus, the algorithm must halt after at most $|f^*|$ iterations, where $|f^*|$ is the maximum flow. Each iteration of this algorithm requires $O(E)$ time, which accounts both for creating the residual graph G_f and for running a search-tree algorithm to find some augmenting path (it doesn't matter what kind of tree-search algorithm; try DFS). Thus, Ford-Fulkerson takes $O(E\,|f^*|)$ time.

It also holds that Ford-Fulkerson will halt if the edge capacities are rational numbers because we can convert any set of rational numbers into a set of integers by multiplying by a sufficiently large integer (do you see why? what integer would do the trick?). However, if any edge capacity is irrational (and represented using infinite precision), then Ford-Fulkerson fails to halt because the algorithm can always find an augmenting path with smaller and small additional flux F, and worse, this infinite sequence of augmentations may not even converge on the maximum flow.[4] This point is not entirely academic: because modern computers represent rational numbers using floating point structures, a careless implementation of Ford-Fulkerson could enter into an infinite loop simply because of a round-off error.

[4] Can you construct an example of such pathological behavior? Uri Zwick identified one such graph with 6 nodes and 9 edges where six edges have some large integer capacity X, two have capacity 1 and one has capacity $\phi_- = (\sqrt{5} - 1)/2 = 0.61803....$

1.7 Edmonds-Karp (1)

We can be smarter about converging on the maximum flow if we are smarter about how we choose which augmenting path to use, if there are multiple choices. One approach, called Edmonds-Karp,[5] is to be greedy about it, i.e., to choose the augmenting path with the largest F. In other words, when given a choice, fill the fattest open pipe first.[6]

The maximum-bottleneck s, t path in a directed graph can be computed in $O(E \log V)$ time using Prim's MST algorithm or Dijkstra's SSSP algorithm: start growing a directed spanning tree S rooted at s by repeatedly finding and adding to S the highest-capacity edge within the cut (S, T) (i.e., among the edges with exactly one vertex in S) until the tree contains t (at which point, the tree must also contain a path from s to t).

The analysis of this algorithm is similar to that of Ford-Fulkerson. A simple upper bound on the running time is to observe that we find, again, at most $|f^*|$ augmenting paths and it takes $O(E \log V)$ time to find each one and update our residual graph data structure. This yields $O(|f^*|\, E \log V)$ time. Note that we can make this running time arbitrarily slow by choosing some edge capacities to be very very large; this changes $|f^*|$, which may not be a desirable property (and could lead to non-polynomial running times).

1.8 Edmonds-Karp (2)

An alternative version of Edmonds-Karp (which was independently discovered by Dinit) chooses the augmenting path with the fewest edges, i.e., to find the shortest open pipe to fill.[7] This version, which I won't go into here,[8] runs in time $O(VE^2)$, which can be improved to $O(V^2E)$ using an insight due to Dinit.

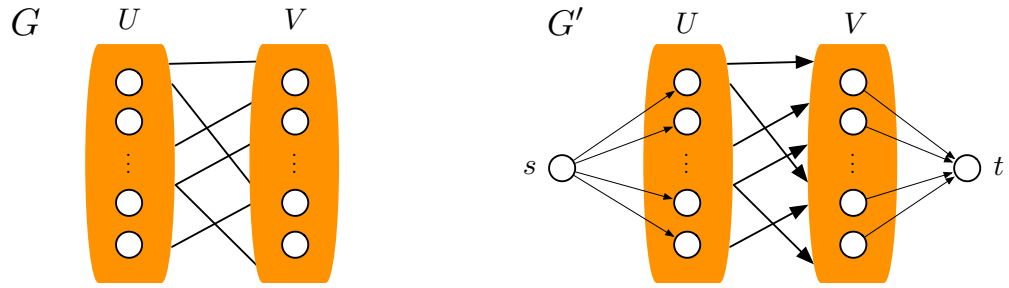
2 Applications

When is finding a max flow or min cut useful? Aside from the obvious applications to flows of commodities (or weapons, or oil, or information, or whatever)[9] through distribution networks, we can also use max flow to solve problems that we can *reduce* to a max flow problem. A reduction is an operation on an input that transforms a problem of type P, for which we have no direct way of solving, into a problem of type Q, for which we do have a way to solve. For instance, suppose we

[5] First published by Yefim Dinic in 1970 and independently published by Jack Edmonds and Richard Karp in 1972.
[6] In the spirit of FIFO, LIFO and OSPF, you might call this "FOPF."
[7] Perhaps "SOPF"?
[8] Briefly: this is like asking for the shortest augmenting path, which means running breadth-first search on the residual graph. Surprisingly, the running time of this approach can be shown to be independent of the edge capacities, which can be a desirable property.
[9] The max-flow/min-cut problem has an interesting history; see Section 4 of these lecture notes

have a weighted bipartite graph and we wish to find a maximum matching, i.e., a subset of edges such that for each edge, neither endpoint is in the same group. We can use a max-flow algorithm to solve this problem.

Here is the reduction to max flow (also shown in the figure below): let G be the given (undirected) bipartite graph with vertex sets U and V, such that every edge $(u, v) \in E$ has $u \in U$ and $v \in V$. We create a new directed graph G' by orienting every edge from U to V, adding two new vertices s and t, adding edges from s to every vertex in U and from every vertex in V to t. Finally, we assign every edge in G' a capacity of 1. Because any matching on G can be transformed into a max flow on G', a solution on G' corresponds to a solution on G (do you see how?).



3 On your own

- In Kleinbert, Tardos, read Chapter 7, Network Flow.

4 The origins of max flow

In *Combinatorial Optimization*, Alexander Schrijver traces the problem of sending flow across a network to the Russian mathematician and economist Leonid Kantorovich[10] in 1939.

The modern work traces back to Ford and Fulkerson (the same pair as above) and their 1954 report *Maximum Flow through a Network* (with followup work in 1956 and 1962). This work, in turn, refers to an original formulation by T.E. Harris and General F.S. Ross (ret.) as a simplified model of railway traffic flow, with a particular application to identifying the minimum cut of the Soviet rail network across Eastern Europe, i.e., the minimum number of lines to bomb in order to disconnect

[10] Kantorovich is also considered the father of linear programming, and won the Nobel Memorial Prize in Economics in 1975. To reflect his many seminal contributions to mathematics, his name is attached to many mathematical properties and theorems.

the Soviet supply stations from the border with Europe.[11] Figure 5 shows a schematic of their rail network, containing 44 vertices and 105 edges. Each edge was given a weight, representing the tonnage of stuff that could be moved between the vertices. The solution of Harris and Ross ("The bottleneck") was not derived by algorithm, but rather by (simulated) trial and error.

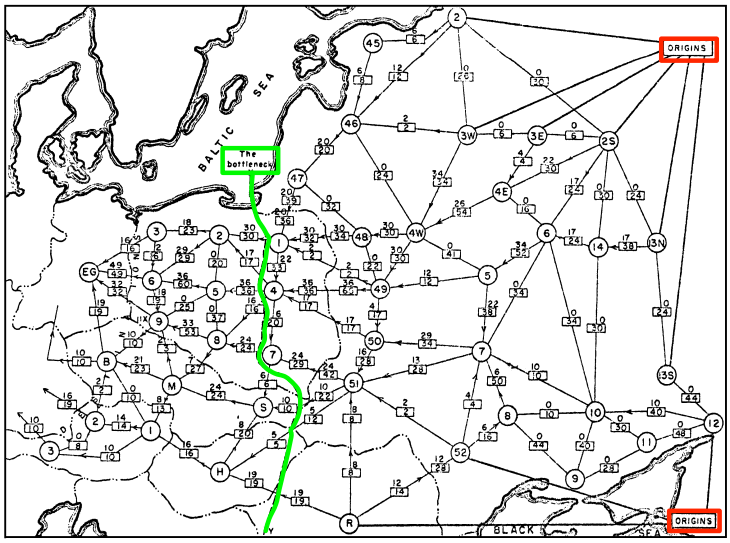


Figure 4: Schematic diagram by Harris and Ross of the railway network of the Western Soviet Union and East European countries, with a maximum flow / cut capacity of 1.63×10^5 tons from Russia (red boxes on right) to Eastern Europe (far left), indicated by the green cut line labeled "The bottleneck." (Adapted from Schrijver's *Combinatorial Optimization*.)

[11] The report by Harris and Ross was originally classified, published in 1955, and then finally declassified in 1999.